Informàtica

ICB0	**Desenvolupament d'aplicacions multiplataforma**
ICC0	**Desenvolupament d'aplicacions web**
M03	Programació
UF6	POO. Introducció a la persistència en BD Accés a BDR-BDOR-BDOO

Diari d'activitats

Isidre Guixà

Curs 2023/2024

Connectivitat via JDBC – Vídeo 01	**14/09/23** **19/09/23**

JDBC és un estàndard de connexió que incorpora Java per connectar amb els SGBDR.

Per poder-lo utilitzar per connectar amb un SGBDR, cal que el fabricant del SGBDR ens faciliti el connector JDBC adequat, que normalment es pot descarregar dels llocs web dels fabricants de SGBDR.

Java incorpora un munt d'interfícies (no classes) en el paquet `java.sql` que nosaltres utilitzarem en els programes que necessitin connexió amb un SGBDR.

- Per compilar els programes, NO necessitem cap driver del fabricant, doncs només necessitem la definició de les interfícies JDBC, que venen incorporades en Java.
- Per executar un programa, el projecte ha de tenir incorporat el connector facilitat pel fabricant del SGBDR i ha de ser coherent amb la versió de Java i la versió del SGBDR.

Informació dels drivers i dades necessàries per connectar amb els "nostres SGBDR habituals":

- Dades per connectar amb Oracle (port habitual: 1521)
  Driver: `ojdbc8_18.3.jar` (per Oracle 18 - descarregable de l'aula del Classroom)
  Driver: `ojdbc8_21.3.jar` (per Oracle 21 - descarregable de l'aula del Classroom)
  URL: `jdbc:oracle:thin:@//IP:PORT/nomBD`
  Classe JDBC: `oracle.jdbc.driver.OracleDriver`
- Dades per connectar amb PostgreSQL (port habitual: 5432)
  Driver: `postgresql-42.2.5.jar` (descarregable de l'aula del Classroom)
  URL: `jdbc:postgresql://IP:PORT/nomBD`
  Classe JDBC: `org.postgresql.Driver`
- Dades per connectar amb MySQL8 (aules Milà) (port habitual: 3306)
  Driver: `mysql-connector-java-8.0.13` (descarregable de l'aula del Classroom)
  URL: `jdbc:mysql://IP:PORT/nomBD`
  Classe JDBC: `com.mysql.cj.jdbc.Driver`

  Si apareix error `java.sql.SQLException: The server timezone value … is unrecognized or represents more than one timezone` cal utilitzar:
  URL: `jdbc:mysql://IP:PORT/nomBD?serverTimezone=UTC`
- Dades per connectar amb MySQL5 (servidor núvol) (port habitual: 3306)
  Driver: `mysql-connector-java-5.1.41-bin_JDJ8-7-6` (aula del Classroom)
  URL: `jdbc:mysql://IP:PORT/nomBD`
  Classe JDBC: `com.mysql.jdbc.Driver`
- Dades per connectar via ODBC (abans Java8):
  Driver: Incorporat en Java (desapareix en Java8)
  URL: `jdbc:odbc:NomDSN`
  Classe JDBC: `sun.jdbc.odbc.JdbcOdbcDriver`

  Pel cas d'ODBC (obsolet) caldrà que la màquina tingui instal·lat el connector ODBC adequat al SGBD a connectar i, en el cas de Windows, cal tenir en compte que distingeix entre connectors ODBC de 32 bits i connectors ODBC de 64 bits.

El protocol JDBC ha anat evolucionant junt amb l'evolució de JAVA.

- En Java6 va aparèixer JDBC4 que incorpora un mecanisme que fa fàcil establir connexió amb un SGBDR.
- La versió estable actual és JDBC4.3

En cas d'haver de fer/retocar un programa en versió Java anterior a JDK6, cal tenir en compte que abans d'intentar establir connexió, cal carregar la classe corresponent al connector del fabricant, amb la instrucció:

`Class.forName(nomDeLaClasseDelConnectorDelFabricant)`

Però el fet d'estar en Java6 o posterior NO implica que ja no calgui carregar la classe via `Class.forName`, doncs dependrà també de si el connector JDBC és 4.x o anterior. Manera simple d'esbrinar-ho és mirar si el connector conté el fitxer `META-INF/services/java.sql.Driver` amb el nom de la classe a carregar en el seu interior.

Informació relativa als projectes de NetBeans que s'adjunten a aquest material:

- Tots estan configurats per a JDK17, que és la versió oficial de Java que utilitzem a l'escola en aquest curs. Tots poden funcionar en JDK8+, però caldrà retocar la configuració del projecte.
- Tots ells, a l'hora d'establir la connexió, incorporen la IP, port, usuari i contrasenya de la màquina on el professor té el SGBD. Cal que l'alumne les canviï per les dades que corresponguin.
- Les dades de connexió (url, usuari, contrasenya) no haurien d'estar mai incorporades en el codi, sinó que cal ubicar-les en fitxer de configuració (en Java s'acostuma a usar un fitxer `properties`).
- Recomanable tenir en el SGBD un esquema/usuari específic per la UF (per exemple, `m03uf6`) per no barrejar amb altres UFs.
- Els exemples usen taules en Oracle generades pel guió `empresaOracleMR.sql` ubicat a Classroom.
- Un projecte que hagi de connectar amb un SGBDR necessita incorporar en el projecte el corresponent connector, que pot residir a qualsevol disc/unitat de la màquina. Evidentment, es diferent la ubicació on el tindrà el professor que la ubicació on el tindrà l'alumne. Per a que no calgui anar a cercar cada vegada les ubicacions, els projectes utilitzen alguna/es de les llibreries:
  - `JDBC Oracle`, que apunta al connector per Oracle
  - `JDBC PostgreSQL`, que apunta al connector per PostgreSQL
  - `JDBC MySQL`, que apunta al connector per MySQL

  Per tant, es recomana tenir aquestes llibreries configurades en el NetBeans i no caldrà retocar ubicacions.

- Projecte `230914_1_ConnexioJDBC_SenseDriverJDBC` permet comprovar que el programa compila però, en executar-lo, es queixa que no troba un driver/connector per la URL indicada.
- Projecte `230914_2_ConnexioJDBC_AmbDriverJDBC` permet comprovar que s'estableix la connexió si el driver/connector és instal·lat en el projecte i la URL és correcta (IP, port, usuari i contrasenya correctes) i el SGBD està engegat i hi ha connectivitat de xarxa.

Establiment de connexió – Gestionar `autoCommit` – Vídeo 1	**19/09/23**

- Projecte `230919_1_ConnexioJDBC_ControlCommit`

Incorpora:

- Com esbrinar l'estat d'`AutoCommit`, que en JDBC sempre ve (en teoria) a `true`.
- Desactivar l'`AutoCommit`.

NO és adequat incorporar URL, usuari, contrasenya... dins el codi. És adequat usar fitxer de propietats.

- Projecte `230921_1_ConnexioJDBC_IndependentDelSGBD`

Les dades de la connexió passen a residir en un fitxer al què s'accedeix abans d'establir la connexió.

Podria ser un fitxer de text amb una sintaxis definida per nosaltres (per exemple, la URL a la primera línia, l'usuari a la segona línia,...) però per facilitar la gestió dels valors a recuperar, en Java s'acostuma a usar els fitxers de propietats (text –habitualment amb extensió `properties`- o XML), ja que disposem de la classe `Properties` per a fer-ne una ràpida recuperació dels valors.

El projecte 230921_1 ha traspassat les dades de connexió (URL, usuari i contrasenya) a un fitxer de propietats anomenat `config.properties` (podria ser qualsevol nom) i, en posar en marxa el programa, via la classe `Properties` s'efectua la recuperació dels valors necessaris per establir la connexió.

El programa del projecte intenta recuperar del fitxer de propietats les propietats `url`, `user` i `pwd` en les variables

`url`, `usuari` i `contrasenya` del programa per després usar-les per establir la connexió.

S'ha comprovat la connexió amb els SGBD Oracle, PostgreSQL i MariaDB de les màquines de l'aula.

Dins el fitxer `config.properties` del projecte es pot veure diverses versions de valors per a les propietats `url`, `user` i pwd, corresponents als 3 SGBD, tot i que només hi pot haver una versió activa i la resta han d'estar comentades.

El programa del projecte també mostra, a títol informatiu, per a la connexió establerta, quin és el nom de la classe que el corresponent fabricant facilita dins el driver per establir la connexió, que és una classe que implementa la interfície `Connection`.

Recuperació d'informació (SELECT) via JDBC – Vídeo 2	**26/09/23**

- Sentència `createStatement` per crear un objecte `Statement` (sentència)
- A través de un objecte `Statement`, obtenir un objecte `ResultSet` via `executeQuery`
- Processar el `ResultSet`, recuperant els valors de cada columna via mètode `get` adequat al tipus de la columna.
- Accés possible a la columna via posició (comptant a partir d'1) o via nom de columna.
- **Alerta** amb l'actuació de `ResultSet.get...(...)` quan la columna a la BD conté valor `null`:
  - Alguns retornen `null`, com `getString(...)`
  - Altres, com els numèrics, retornen 0 i... evidentment un `null` no és igual a ZERO!!!

  El programador ha de ser conscient d'aquest fet i, quan vulgui saber si una columna conté valor `null`, després d'efectuar la lectura (via `get`), disposa del mètode `ResultSet.wasNull()` per esbrinar si la **darrera** lectura corresponia a valor `null`.

Exercici per lliurar a Classroom (tasca 1):

Projecte amb 2 programes:

- `P01_MostrarDepartaments`, que mostri els departaments de l'empresa en format:

```
CODI DNOM           LOC
---- -------------- --------------
  10 COMPTABILITAT  SEVILLA
  20 INVESTIGACIÓ   MADRID
  30 VENDES         BARCELONA
  40 PRODUCCIÓ      BILBAO
```

- `P02_MostrarEmpleats`, que mostri la següent informació dels empleats de l'empresa, ordenats per cognom:

```
CODI COGNOM     DNOM
---- ---------- --------------
7876 ALONSO     INVESTIGACIÓ
7499 ARROYO     VENDES
7782 CEREZO     COMPTABILITAT
7902 FERNANDEZ  INVESTIGACIÓ
7788 GIL        INVESTIGACIÓ
7566 JIMENEZ    INVESTIGACIÓ
7900 JIMENO     VENDES
7654 MARTIN     VENDES
7934 MUÑOZ      COMPTABILITAT
7698 NEGRO      VENDES
7839 REY        COMPTABILITAT
7521 SALA       VENDES
7369 SANCHEZ    INVESTIGACIÓ
7844 TOVAR      VENDES
```

JDBC – Compte amb valors Date i numèrics null	**28/09/23**

- JDBC gestiona els camps data amb la classe `java.sql.Date` enlloc de la classe `java.util.Date` i cal tenir present que `java.sql.Date` deriva de `java.util.Date`.

- En recuperar un camp numèric (via qualsevol mètode `get`), Java dona valor 0 tant si el valor a la BD és 0 com si és `null` i en molts casos interessa distingir la situació. Per aconseguir-ho, Java facilita el mètode `ResultSet.wasNull()` que permet saber si el darrer valor recuperat amb get era `null` o no. Per tant, si es coneix que un camp numèric X admet `null`, si es vol tenir la seguretat del valor recuperat via `get` caldria fer:

```
tipus v = rs.getTipus();
if (rs.wasNull()) {
  // v val zero però sabem que en realitat a la BD era null
} else {
  // en cas que v valgui zero, tenim la seguretat que a la BD hi havia un zero
}
```

En mostrar informació provinent de la BD, cal fer una gestió acurada de tots els camps que poden tenir valor NUL, encara que el joc de dades no en tingui:
- En els camps numèrics és necessari per què acostuma a no ser vàlid mostrar 0 quan en realitat és valor nul.
- En la resta de camps, no ens podem permetre la llicència de mostrar el mot `null`.

Exercici per lliurar a Classroom (tasca 2):

Projecte amb programa que mostri la següent informació de tots els empleats:

```
CODI COGNOM     DEPARTAMENT    CAP        DATA ALTA OFICI    SALARI   COMISSIÓ
---- ---------- -------------- ---------- --------- -------- -------- --------
```

on:
- Columna `DEPARTAMENT` ha de mostrar el nom del departament
- Columna `CAP` ha de mostrar el cognom del cap
- Columna `DATA ALTA` ha de mostrar la data en format dd/mm/yyyy

Exercici per lliurar a Classroom (tasca 3):

Mostrar per la consola, la informació de totes les comandes, amb les seves línies, ordenades per número de comanda i les línies per número de línia, en format similar a:

```
Comanda:
        Número: xxx
        Data Comanda: xxx (format dd/mm/yyyy)
        Tipus:
        Data Tramesa: xxx (format dd/mm/yyyy)
        Client: Codi – Nom
        Import total: xxx
Línies:
        Número: xxx
                Producte: Codi – Descripció
                Qtat: xxxx – Preu: xxxx – Import: xxxx
        Número: xxx
                Producte: Codi – Descripció
                Qtat: xxxx – Preu: xxxx – Import: xxxx
        ...
Comanda: ...
```

Versió 1 a desenvolupar: Usar una única `SELECT` que recuperi comandes i línies.

Caldrà recórrer el `ResultSet` per poder mostrar les línies, però totes les línies d'una mateixa comanda contenen la informació de la comanda i aquesta només ha de sortir abans de la primera línia. Per tant, caldrà un semàfor per controlar quan es canvia de comanda i, en aquest cas, mostrar les dades de la comanda.

Tingueu present que podria donar-se el cas d'existir alguna comanda sense línies, i també ha d'aparèixer.

Exercici per lliurar a Classroom (tasca 4):

Versió 2 del programa anterior: Usar dues `SELECT`:

- `SELECT` que obtingui la capçalera de la comanda
- `SELECT` que obtingui les línies de detall de cada comanda

Cal fer un recorregut per les comandes obtingudes en la primera `SELECT` i per a cada comanda cal executar la segona `SELECT` per obtenir les línies de detall de la corresponent comanda.

Atenció:

- Per executar la primera `SELECT` necessitarem un `Statement` (suposem `staCom`) pel que executarem la SELECT i obtindrem un `ResulSet` (suposem `rsCom`) que anirem recorrent.
- Donada una comanda, necessitem executar una altra `SELECT`, que precisa d'un `Statement`. **NO** es pot usar el mateix `staCom` de la primera `SELECT`, doncs provocaria el tancament del `ResultSet rsCom` i no podríem continuar recorrent `rsCom`.

És a dir, cada `ResultSet` amb el seu `Statement`, doncs tots 2 treballen simultàniament

Instruccions parametritzades per efectivitat i per evitar injecció de codi – Vídeo	**17/10/23**

En el vídeo s'explica com efectuar una consulta parametritzada i l'autor explica la importància d'utilitzar instruccions parametritzades per evitar injecció de codi. En realitat hi ha dos motius per usar sentències parametritzades:

- Millorar eficiència

  Suposem una consulta per recuperar nom i sou dels empleats d'un departament indicat pel seu codi, que pot anar canviant i que té en una variable `dept` (suposem numèrica)

  Un usuari inexpert podria pensar en dissenyar la consulta següent:

  ```
  String s = "select nom, sou from emp where id_dept = " + dept; (1)
  ```

  En cas que la variable `dept` fos cadena, caldria tancar-la entre cometes simples a la consulta:

  ```
  s = "select nom, sou from emp where id_dept = '" + dept + "'"; (2)
  ```

  Cada vegada que volgués obtenir la informació, executaria (suposant que `st` és `Statement`)

  ```
  Query q = st.executeQuery(s);
  ```

  Suposem que executa la consulta diverses vegades havent canviat el valor de `dept` per 10, 20, 30.

  El SGBD hauria rebut les consultes:

  ```
  select nom, sou from emp where id_dept = 10;
  select nom, sou from emp where id_dept = 20;
  select nom, sou from emp where id_dept = 30;
  ```

  Quan un SGBD rep una instrucció, l'analitza, mira que sigui correcte i crea el pla d'execució (programa intern) per poder efectuar el que demana la instrucció. Per millorar eficiència, es guarda la instrucció i el pla d'execució, de manera que cada vegada que arriba una instrucció, mira si la té guardada i si la té (ha d'estar exactament escrita) no cal que l'analitzi ni que creï el pla d'execució, per què ja ho te fet i pot passar directament a l'execució del pla.

  Però en el cas anterior, **les tres instruccions que "fan el mateix", no són idèntiques**, doncs canvia el valor 10, 20,... i ha de fer l'anàlisi i elaborar el pla d'execució cada vegada.

  Treballar amb sentències parametritzades evita el problema, per que el què arriba al SGBD és una sentència parametritzada, la qual analitza i per a la que elabora el pla d'execució, un sola vegada! Cada vegada que es demana l'execució, el SGBD rep el valor dels paràmetres, que incorpora al pla d'execució i executa. **Estalvi bestial de feina pel SGBD**!

- Evitar injecció de codi (hi ha molts vídeos a la web que ho expliquen)

La injecció de codi només pot passar quan construïm sentències SQL (com en el cas anterior) via concatenació de text i valors de variables cadena, les quals han estat emplenades per un usuari amb vocació de hacker.

- Cas de la sentència (1) anterior on la variable `dept` és numèrica:

Imaginem que un programa demana per pantalla que l'usuari introdueixi el codi de departament i que el seu valor va directament a variable `dept`. Si l'usuari introdueix `10`, `20`, `30`..., cap problema de seguretat doncs en aquest cas (1) executa les consultes següents:

```
select nom, sou from emp where id_dept = 10;
select nom, sou from emp where id_dept = 20;
select nom, sou from emp where id_dept = 30;
```

Però... I si l'usuari és hacker i introdueix, per exemple: `10 or 1=1`

fixeu-vos que en construir la sentencia (1) queda:

```
select nom, sou from emp where id_dept = 10 or 1=1
```

i en conseqüència, accedeix a les dades dels usuaris de tots els departaments!!! **Ha injectat codi!!!**

Amb una sentència parametritzada (a més de ser més eficient), això no passaria:

```
s = "select nom, sou from emp where id_dept = ?";
PreparedStatement ps = connection.prepareStatement(s);
ps.setByte(1,dept);
```

En cas que l'usuari hagi introduït `10 or 1=1`, es produirà una excepció en executar `setByte` i no es produirà injecció de codi.

- Cas de la sentència (2) anterior on la variable `dept` és cadena:

Imaginem que un programa demana per pantalla que l'usuari introdueixi el codi de departament i que el seu valor va directament a variable `dept`. Si l'usuari introdueix `10`, `20`, `30`..., cap problema de seguretat doncs en aquest cas (1) executa les consultes següents:

```
select nom, sou from emp where id_dept = '10';
select nom, sou from emp where id_dept = '20';
select nom, sou from emp where id_dept = '30';
```

Però... I si l'usuari és hacker i introdueix, per exemple: `10' or '1'='1`

fixeu-vos que en construir la sentencia (2) queda:

```
select nom, sou from emp where id_dept = '10' or '1'='1'
```

i en conseqüència, accedeix a les dades dels usuaris de tots els departaments!!! **Ha injectat codi!!!**

Amb una sentència parametritzada (a més de ser més eficient), això no passaria:

```
s = "select nom, sou from emp where id_dept = ?";
PreparedStatement ps = connection.prepareStatement(s);
ps.setString(1,dept);
```

En executar la sentència, si l'usuari ha introduït `10' or '1'='1` a la variable `dept`, el SGBD rep aquest valor i l'incorpora com a `String` a la instrucció SQL (canviant `'` per `''`) i executarà:

```
select nom, sou from emp where id_dept = '10'' or ''1''=''1'
```

sense obtenir cap resultat. **Hem evitat la injecció de codi!!!**

i en conseqüència, accedeix a les dades dels usuaris de tots els departaments!!! **Ha injectat codi!!!**

- Projecte `231017_1_ProvaInjeccioCodi`: Usa una consulta amb dada a introduir per l'usuari i si l'usuari escriu `10 or 1=1`, mostra la informació de tots els departaments. Injecció de codi! Executeu `Prova...`

- Projecte `231017_2_ProvaConsultaParametritzada`: Projecte anterior retocat amb consulta parametritzada. Executeu `Prova...`

Exercici per lliurar a Classroom (tasca 5):

Refer el projecte de la tasca 4 usant sentència parametritzada allà on calgui.

Instruccions NO SELECT (DML-DDL-DCL) via JDBC	**19/10/23**

- **Recordem** que per executar instruccions `SELECT`, hem d'utilitzar el mètode `executeQuery`:

- Per a consultes no parametritzades:

```
Statement st = Connection.createStatement();
Resulset rs = st.executeQuery(consultaSelect);
```

- Per a consultes parametritzades:

```
PreparedStatement ps = Connection.prepareStatement(consultaAmbParàmetres);
```

Abans d'executar:
- Emplenem els paràmetres amb mètodes `set`
- Els paràmetres s'indiquen per posició a partir de 1.

Per executar: `Resultset rs = ps.executeQuery();`

En els dos casos, el resultat és un `ResulSet` que hem de processar.

- **Les instruccions SQL NO SELECT (DML-DDL-DCL) s'executen amb mètodes `executeUpdate`.**

- Per a instruccions NO SELECT no parametritzades:

```
Statement st = Connection.createStatement();
int i = st.executeUpdate(instrucció)
```

- Per a instruccions parametritzades:

```
PreparedStatement ps = Connection.prepareStatement(instruccióAmbParàmetres);
```

Abans d'executar:
- Emplenem els paràmetres amb mètodes `set`
- Els paràmetres s'indiquen per posició a partir de 1.

Per executar: `int i = ps.executeUpdate();`

En els dos casos, el resultat és un `int` que ens indica:

- El número de files processades en cas d'instrucció DML (`INSERT` – `DELETE` – `UPDATE`)
- Zero si es tracta d'una instrucció DDL (create/alter table/index...) o DCL (create/drop user, grant, revoke)

Atenció amb la informació que JDBC dona respecte commit/rollback:

It is strongly recommended that an application explicitly commits or rolls back an active transaction prior to calling the close method. If the close method is called and there is an active transaction, the results are implementation-defined.

És a dir, si intentem tancar una connexió amb canvis pendents de fer `commit/rollback`, JDBC no assegura què succeirà; ho deixa a decisió del fabricant del connector, que pot decidir:

- Generar excepció
- Fer `rollback` automàtic (sembla que seria el més normal)
- Fer `commit` automàtic

- Oracle, en la documentació del seu connector `ojdbc8.jar`, diu:

  If the auto-commit mode is disabled and you close the connection without explicitly committing or rolling back your last changes, then an implicit COMMIT operation is run.

- SQLServer, en la documentació diu:

  Calling the close method in the middle of a transaction causes the transaction to be rolled back.

- PostgreSQL & MySQL, via comprovació (no trobat en documentació), executen `rollback` en tancar connexió

Davant el fet que no tots els SGBD actuen de la mateixa manera, és altament recomanable, per no dir imprescindible que abans de tancar una connexió, efectuar `rollback`.

Exercici per dijous, 26 d'octubre (tasca 6)

Desenvolupar programa que afegeixi a la taula PRODUCTE la columna DARRER_PVP i que l'empleni, per a tots els productes de la taula, amb el darrer preu de venda. Aquesta columna ha de permetre valors nuls, doncs és possible que un producte encara no hagi tingut cap venda.

El programa ha d'informar de qualsevol problema que sorgeixi. Així, per exemple, si executeu el programa quan la columna ja existeix, ha d'informar d'aquest fet i avortar l'execució. És clar, que per provar el programa diverses vegades, haureu d'eliminar la columna des d'una connexió a la BD via *SQLDeveloper* o *SQL*Plus*.

Solució – Es pot fer dues versions:

- Versió 1:
  1. Afegeix la darrera columna amb un `executeUpdate` d'instrucció no parametritzada.
  2. Recupera tots els productes en `ResultSet` amb un `executeQuery` de SELECT no parametritzada
  3. Per cada producte calcula el darrer preu de venda amb un `executeQuery` de SELECT parametritzada.
     Cal tenir en compte que aquesta SELECT pot retornar una fila (si el producte s'ha venut alguna vegada) o cap fila (si el producte no s'ha venut mai).
     1. Si el producte s'havia venut alguna vegada, actualitza el darrer preu de venda amb un `executeUpdate` d'instrucció parametritzada.
     2. Si el producte no s'havia venut mai, no emplena res i la nova columna quedarà amb `NULL`.

- Versió 2, on després d'afegir la columna, amb una única instrucció `update`, modifica tots els productes.
  Evidentment aquesta segona versió és més eficient, doncs amb una única instrucció informem al SGBD què ha de fer per aconseguir emplenar la nova columna.
  No sempre serà possible assolir l'objectiu amb una única instrucció, però si ho és, per què usar-ne més?

Exercici per dimarts, 31 d'octubre (tasca 7)

Desenvolupar programa pensat per intentar eliminar clients, amb el següent funcionament repetitiu

- Demani a l'usuari per consola un codi de client, amb missatge:
  `Introdueixi codi de client (zero per finalitzar el programa)`
  El programa ha d'obligar que el valor introduït per l'usuari no sigui negatiu
- Si el valor és zero, finalitza el programa.
- Si el valor no és zero (serà estrictament positiu), ha d'invocar mètode `eliminarClient` encarregat de cercar i eliminar el client indicat.

Respecte el mètode `eliminarClient` ha de controlar totes les possibilitats.

- Que el codi de client sigui vàlid (estrictament positiu), cosa que en el nostre cas no passarà mai, doncs es suposa que el programa ja ho controla, però ho programem per si algun dia usem el mètode en un altre programa.
  Si no és vàlid, ha de generar una `RuntimeException` amb missatge adequat.
- Que el client a eliminar existeixi.
  Si no existeix, ha de generar una `RuntimeException` amb missatge adequat.
- Que el client es pugui eliminar.

Si no es pot eliminar, ha de generar una `RuntimeException` amb missatge adequat.

El mètode no ha de mantenir cap interacció amb l'usuari. O tot va bé, o si hi ha algun problema ho comunicarà via `RuntimeException`.

El programa principal, en invocar el mètode `eliminarClient`, ha d'estar preparat per esbrinar si tot ha anat bé o si s'ha produït algun problema i informarà a l'usuari de la correcta eliminació o del problema sorgit.

Observació:

És molt probable que en el desenvolupament del mètode `eliminarClient` penseu en executar dues instruccions SQL:
1a. Instrucció `select` per esbrinar si el client existeix.
2a. Instrucció `delete` per intentar eliminar el client, dins un `try {... } catch (SQLException ex)` per controlar el possible error per violació de FK.

Però si ho penseu millor, fixeu-vos que us podeu estalviar la primera `select`, doncs en executar la instrucció `delete`, podeu comprovar el valor que retorna (número de files eliminades), de manera que un zero indica que no existia el client. Cal sempre cercar el camí més eficient!

Aprofundiment en `Statement`/`PreparedStatement` sota mateixa/diferent connexió	**31/10/23**

Exemples de l'efecte d'utilitzar diferents `Statement`/`PreparedStatement` amb una mateixa connexió i amb diferents connexions.

Projecte `231031_1_JDBC_Statement`

- Programa P01 – 2 `Statement` en mateixa connexió:

  Ambdós `Statement` comparteixen les dades sense necessitat de fer `commit` per a que cada `Statement` vegi el què ha fet l'altre...

  Tenim `st1` i `st2`. Fem un `INSERT` (empleat 777-Gotera) via `st1` i posteriorment recuperem el contingut de la taula utilitzant `st1` i també amb `st2` i en les dues recuperacions observem la fila inserida.

- Programa P02 – `Statement` en diferents connexions.

  En aquest cas es demostra que no comparteixen les dades. Cal fer `commit` per a que un vegi les modificacions efectuades per l'altre.

  Tenim `st1` i `st2`. Fem un `INSERT` (empleat 8888-Pepeillo) via `st1` i posteriorment recuperem el contingut de la taula utilitzant `st1` i també amb `st2` i només observem la fila inserida en la recuperació efectuada via `st1`. Si voleu, podeu retocar el programa afegint un `con1.commit()` abans d'executar la recuperació de les files via `st2` i llavors ja hi observareu la fila inserida via `st1`.

Els dos programes mostren l'efecte sobre `Statement`, però el mateix efecte té lloc en `PreparedStatement`

Aprofundiment en gestió de `ResultSet`: Modificar files i tirar enrere	**31/10/23**

Fins ara, hem utilitzat els `ResultSet` per recollir resultats d'instruccions SELECT I els hem recorregut cap endavant. Aquesta és la utilització més bàsica, però `ResultSet` permet més accions.

Si es vol disposar de `ResultSet` més "funcionals", s'ha d'indicar en el moment de crear l'`Statement` o el `PreparedStatement` sobre el què s'executarà la consulta i s'obtindrà el `ResultSet`.

- Hi ha un mètode `createStatement` que permet incorporar dos paràmetres molt interessants:

```
Statement createStatement(int resultSetType,
                          int resultSetConcurrency)
                   throws SQLException
```

on:
`resultSetType` pot valer:

`ResultSet.TYPE_FORWARD_ONLY` (només endavant – per defecte)
`ResultSet.TYPE_SCROLL_INSENSITIVE` (endavant i enrere)
`ResultSet.TYPE_`**`SCROLL_SENSITIVE`** (endavant i enrere)

`resultSetConcurrency` pot valer:
`ResultSet.CONCUR_READ_ONLY` (no es pot modificar les files recorregudes – per defecte)
`ResultSet.CONCUR_UPDATABLE` (permet modificar les files recorregudes, eliminar i inserir)

- Hi ha un mètode `prepareStatement` que permet incorporar els mateixos paràmetres del mètode anterior:

```
PreparedStatement prepareStatement(String sql,
                                   int resultSetType,
                                   int resultSetConcurrency)
                            throws SQLException
```

- Diferència entre `TYPE_SCROLL_SENSITIVE` i `TYPE_SCROLL_INSENSITIVE`:

Table 17-1 Visibility of Internal and External Changes for Oracle JDBC

Result Set Type	Can See Internal DELETE?	Can See Internal UPDATE?	Can See Internal INSERT?	Can See External DELETE?	Can See External UPDATE?	Can See External INSERT?
forward-only	no	yes	no	no	no	no
scroll-sensitive	yes	yes	no	no	yes	no
scroll-insensitive	yes	yes	no	no	no	no

Atenció! Per a que un `ResultSet` sigui modificable, cal haver:
- Creat el `Statement` o el `PreparedStatement` adequadament (dient-li `CONCUR_UPDATABLE`) i, a més,
- NO es pot fer "select * " (cal indicar el nom de les columnes a recuperar)
- Només sobre una taula

Més informació sobre el connector JDBC d'Oracle: Documentació d'Oracle 18c – Documentació d'Oracle 21c

Projecte exemple: `231031_2_JDBC_ResultSet_CRUD`, que mostra:

- Com modificar camps de la fila on està aturat el cursor que recórrer el `ResultSet`.
  - `rs.updateTipus(nomColumna, nouValor)`, per modificar el contingut d'una columna
  - Quan ja s'han fet totes les modificacions dels camps de la fila, cal executar `rs.updateRow()`, moment en el que el programa envia la instrucció `update` de les columnes afectades a la BD. Evidentment, si s'infringeix alguna restricció PK. FK, CK, UN o NN, es produirà una `SQLException`.

  El BUCLE1 dins el programa exemple mostra com es modifica el nom de tots els departaments

- Com s'afegeix una fila (cal "moure" el cursor a una fila especial per inserir, de manera similar a quan utilitzem una interfície gràfica d'accés a BD en la que per afegir una fila ens hem de situar a la darrera fila de la graella mostrada per pantalla)
  - `rs.moveToInsertRow()`, per "crear" una nova fila en el lloc on està ubicat el cursor que recorre el `ResultSet`.
  - `rs.updateTipus(nomColumna, nouValor)`, per anar emplenant les columnes
  - Quan ja s'han emplenat tots els camps de la fila, cal executar `rs.insertRow()`, moment en el que el programa envia la instrucció `insert` a la BD. Evidentment, si s'infringeix alguna restricció PK. FK, CK, UN o NN, es produirà una `SQLException`.

  El BUCLE2 dins el programa exemple afegeix departaments (del 61 al 63)

- Com eliminar la fila on està aturat el cursor que recorre el `ResultSet`.
  - `rs.deleteRow()` és la instrucció que elimina la fila on està ubicat el cursor, moment en el que el programa envia instrucció `delete` a la BD. Evidentment, si s'infringeix alguna restricció PK, FK, CK, UN o NN, es produirà una `SQLException`.

  En el PUNT3 del programa exemple, s'elimina el departament 40 (que no té empleats i per això es pot eliminar i no es genera cap `SQLException`).

- Com tornar a recórrer el `ResultSet`, passant a l'inici amb el mètode `rs.beforeFirst()`.
- També hi ha mètodes `first()`, `last()`, `previous()`, `rowRefresh()`. Mirar la documentació de `ResultSet`.

Atenció! El programa modifica noms, afegeix departaments i elimina un departament i fa `commit`... Per tant, la BD queda modificada i si el torneu a executar, us petarà la inserció (per PK). Haureu de tornar a executar el guió que deixa les taules amb les dades inicials.

Exercici per dimarts, 7/11 (tasca 8) – Es disposa de la classe de dijous 2, per resoldre dubtes al respecte

Desenvolupar programa que, en iniciar, es connecti a la BD on hi hagi les taules de l'esquema `SANITAT` (teniu el guió a l'aula de Classroom), carregui tots els malalts en un `ResultSet` i mostri un menú (per consola) amb opcions:

1. Mostrar malalts => Ha d'invocar mètode `mostrarMalalts()`.
2. Cercar malalt per inscripció => Ha d'invocar mètode `cercarMalaltPerInscripcio()`
3. Cercar malalts per part de cognom => Ha d'invocar mètode `cercarMalaltsPerPartDeCognom()`
4. Refrescar dades => Ha d'invocar mètode `refrescarResultSet()`
0. Sortir

A continuació es detalla el funcionament que ha de tenir cada mètode.

Si creieu que algun(s) mètode(s) ha(n) de tenir algun(s) paràmetre(s), incorporeu-los.

Si creieu que és interessant afegir algun altre mètode adequat per ser invocat des de diversos mètodes, afegiu-lo.

- Mètode `mostrarMalalts()`
  Ha de fer un recorregut pel `ResultSet` i mostrar els malalts existents, en format:
  `INSCRIPCIÓ – COGNOM - DOMICILI – DATA NAIX. – SEXE – NSS`
  amb columnes ben justificades i on la data de naixement es mostri en format `dd/mm/yyyy`.
  Si no hi ha cap malalt, cal informar.
  En finalitzar, es torna al menú inicial.

- Mètode `cercarMalaltPerInscripcio()`
  Ha de demanar per consola un codi de malalt.
  El mètode ha de controlar que sigui un codi vàlid.
  Si no és codi vàlid, s'informa i es surt del mètode.
  Si és codi vàlid, es cercarà dins el `ResultSet`.
  Si no existeix, s'informa i es surt del mètode.
  Si existeix, cal mostrar-lo en format:
  `INSCRIPCIÓ – COGNOM - DOMICILI – DATA NAIX. – SEXE – NSS`
  amb columnes ben justificades i on la data de naixement es mostri en format `dd/mm/yyyy`.
  En finalitzar, es torna al menú inicial.

- Mètode `cercarMalaltsPerPartDeCognom()`
  Ha de demanar per consola part de cognom de malalt.
  El mètode ha de controlar que sigui valor vàlid.
  Si no és valor vàlid, s'informa i es surt del mètode.
  Si és valor vàlid, es cercarà dins el `ResultSet` els malalts que el seu cognom contingui la part introduïda per l'usuari, insensible a majúscules-minúscules.
  Si no existeix, s'informa i es surt del mètode.
  Si es troba (poden ser varis malalts), cal mostrar-los en format:
  `INSCRIPCIÓ – COGNOM - DOMICILI – DATA NAIX. – SEXE – NSS`
  amb columnes ben justificades i on la data de naixement es mostri en format `dd/mm/yyyy`.
  En finalitzar, es torna al menú inicial.

- Mètode `refrescarResultSet()`
  Ha de tornar a carregar les dades dels malalts, per si hi ha hagut alguna modificació a la BD.

Observacions:

- La càrrega inicial del `ResultSet` ha de fer exactament el mateix que el mètode `refrescarResultSet`. Per tant, una vegada establerta la connexió, invoquem `refrescarResultSet` per aconseguir la primera càrrega.
- Degut a que el `ResultSet` s'ha de recórrer vàries vegades, necessitem poder-nos ubicar al principi. Per aconseguir-ho, cal declarar l'`Statement` a partir del qual s'executa la consulta que obté el `ResultSet`, com:
```
con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_READ_ONLY);
```
El segon paràmetre el podem posar de "només lectura" per què en el programa demanat, no hem de fer cap modificació a través del `ResultSet`.
- És adequat crear un objecte `SimpleDateFormat` global amb el format que se'ns demana per les dates, per usar-lo allà on faci falta i no haver-lo de crear cada vegada.
- El `ResultSet` es pot crear amb els malalts ordenats per inscripció (no es demana) però així, en `cercarMalaltPerInscripcio` podem aprofitar que el `ResultSet` està ordenat per inscripció per fer una cerca seqüencial en dades ordenades, ja que quan passem del codi d'inscripció que estem cercant, no cal continuar, doncs ja es té la seguretat de que no existirà la inscripció cercada.
- En `cercarMalaltPerPartDeCognom` no ens podem aprofitar de que el `ResultSet` està ordenat per inscripció.

Exercici per dimarts, 14/11 (tasca 9) – Es disposa de la classe de dijous 9, per resoldre dubtes al respecte

Ampliar programa anterior efectuant altes, baixes i modificacions de malalts via el `ResultSet`. Concretament:

5. Inserir malalt => Ha d'invocar mètode `inserirMalalt()`.
6. Eliminar malalt per inscripció => Ha d'invocar mètode `eliminarMalaltPerInscripcio()`
7. Eliminar malalts per part de cognom => Ha d'invocar mètode `eliminarMalaltsPerPartDeCognom()`
8. Modificar malalt => Ha d'invocar mètode `modificarMalalt()`
9. Validar els canvis => Ha de fer `commit`
10. Desfer els canvis => Ha de fer `rollback`

A continuació es detalla el funcionament que ha de tenir cada mètode.

- Mètode `inserirMalalt()`:
  Ha de demanar totes les dades d'un malalt.
  Només ha de comprovar que les dades són correctes per cada camp, però no cal que comprovi restriccions de PK, FK, CK, UN i NN.
  Cal efectuar la inserció via el `ResultSet`, informant si s'ha pogut efectuar la inserció a la BD o si pel contrari s'ha violat alguna restricció, informant en aquest cas del problema (missatge que retorna Oracle).
- Mètode `eliminarMalaltPerInscripcio()`
  Ha de fer un tractament similar al `cercarMalaltPerInscripcio()`, però tenint en compte que, si no existeix en el `ResultSet`, s'informa de la no existència i si existeix, cal eliminar-lo (via `ResultSet`), informant si s'ha pogut eliminar de la BD o si pel contrari s'ha violat alguna restricció, informant en aquest cas del problema (missatge que retorna Oracle).
- Mètode `eliminarMalaltsPerPartDeCognom()`
  Ha de fer un tractament similar al `cercarMalaltsPerPartDeCognom()`, però tenint en compte que, si no existeix cap malalt en el `ResultSet`, s'informa de la no existència i si existeixen, cal eliminar-los (via `ResultSet`), informant si s'han pogut eliminar de la BD o si pel contrari s'ha violat alguna restricció, informant en aquest cas del problema (missatge que retorna Oracle).
- Mètode `modificarMalalt()`:
  Ha de demanar un número d'inscripció de malalt. L'ha de cercar.
  Si no existeix, s'informa.
  Si existeix, ha de mostrar submenú:
  1. Cognom: `<cognom>`
  2. Domicili: `<domicili>`
  3. Data Naix: `<data naixement>`
  4. Sexe: `<sexe>`

5. NSS: `<nss>`
0. Finalitzar

de manera que l'usuari pugui modificar les dades del(s) camp(s) que vulgui tantes vegades com vulgui
En finalitzar, cal traspassar els canvis a la BD (via `Resultset`), informant si s'ha pogut efectuar la modificació a la BD o si pel contrari s'ha violat alguna restricció, informant en aquest cas del problema (missatge que retorna Oracle).

Observacions:

- Per poder modificar dades de la BD via un `ResultSet`, aquest no pot ser de només lectura, com havíem fet en versió anterior. Ens cal retocar l'`Statement` a partir del qual s'obté el `ResultSet`, declarant-lo:

```
con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_UPDATABLE);
```

- Quan tingueu desenvolupat el mètode `inserirMalalt`, comproveu els següents funcionaments:
  - Posar en marxa el programa.
  - Sense *Mostrar Malalts*, executeu una inserció de malalt amb dades que no infringeixin cap de les restriccions pels diversos camps.
  - Posteriorment executeu *Mirar Malalts*. Veureu que el nou malalt està en primera posició. Lògic, doncs no ens havíem mogut pel `ResultSet`.
  - Executeu una nova inserció de malalt, que no provoqui cap error.
  - Torneu a executar *Mirar Malalts*. Veureu que el nou malalt està en darrera posició. Lògica, doncs en el primer *Mirar Malalts*, havíem fet un recorregut i ens havíem quedat al final de tots. Queda clar que `rs.moveToInsertRow` insereix la nova fila en la posició on és el cursor que recorre el `ResultSet`.
  - Si comproveu el contingut de la taula `MALALTS` des de SQLDeveloper, no apareix cap dels 2 nous malalts, doncs encara no s'ha validat els canvis.
  - Executeu opció "`commit`" i comproveu que dins la taula `MALALTS` des de SQLDeveloper ja apareixen els nous malalts.
  - Crear un malalt amb un codi d'inscripció ja existent. Observeu que, en el moment d'executar `insertRow` és quan "peta", informant que `s'ha infringit la restricció única (MALALT_PK)`

Exercici per les properes classes

Crear projecte Java `P1_Model` amb classe `Pais`, amb:
- `codiIso`: cadena(2) – obligatori amb 2 caràcters – majúscules
- `nomPais`: cadena(40) – obligatori no buit
- Mètodes habituals: constructor, getter-setter, `toString`
- Considerar que 2 països són iguals si tenen mateix `codiIso`

Crear en Oracle taula `Pais` amb columnes:
- `codi`: varchar2(2) – PK
- `nom`: varchar2(40) – not null – unique

Crear projecte Java `P2_GestioBD` amb programa `main` que estableixi connexió amb la BD Oracle on hi ha la taula `Pais` i mostri menú amb opcions:
1. Afegir país => Ha d'invocar mètode `afegirPais`
2. Mostrar països => Ha d'invocar mètode `mostrarPaisos`
3. Cercar país => Ha d'invocar mètode `cercarPais`
4. Modificar país => Ha d'invocar mètode `modificarPais`
5. Eliminar país => Ha d'invocar mètode `eliminarPais`
6. Enregistrar canvis => Ha d'invocar mètode `validarCanvis`
7. Desfer canvis => Ha d'invocar mètode `desferCanvis`
0. Sortir

Ubicar TOTS els mètodes que accedeixen a la BD en una classe separada de la classe on hi ha el programa. És a dir, el projecte `P2_GestioBD` ha de tenir 2 classes:

- `Programa`, on hi hagi el `main`, el menú i els diversos mètodes que a invocar des de cada opció de menú
- `GestorBD`, on hi hagi tots els mètodes que accedeixen a la BD.

Exercici per dijous, 16/11 (tasca 10)

- Desenvolupar classe `GestorBDException extends Exception` per totes les excepcions que es puguin generar en els mètodes de `GestorBD`.

- Desenvolupar `GestorBD` amb mètodes següents, que en cas d'error han de generar una `GestorBDException`. **MAI** han de donar error per pantalla. **MAI** han de tenir interacció amb l'usuari.
  - `public Connection establirConnexióBD()`, que retorni la connexió a la BD
  - `public boolean existeixPaisBD (Connection con, String codiPais)`, que comprovi si existeix el país indicat.
  - `public void afegirPaisBD(Connection con, Pais p)`, que intenti inserir el país a la BD
  - `public void validarCanvisBD (Connection con)`, que faci `commit`
  - public void `desferCanvisBD (Connection con)`, que faci `rollback`

- Desenvolupar `Programa` amb `main` que:
  - Ha de tenir un objecte `Connection` per mantenir la connexió, que ha d'emplenar en començar invocant el mètode `establirConnexióBD`.
  - Ha de crear menú indicat i mètodes indicats amb codi `/* TO DO */` inicialment.
  - Programar mètode `validarCanvis` que invoqui `validarCanvisBD`
  - Programar mètode `desferCanvis` que invoqui `desferCanvisBD`
  - Programar Mètode `afegirPais`
    Ha de demanar un codi de país –controlant que sigui vàlid i transformant-lo en majúscules-.
    Ha de comprovar si existeix o no a la BD, invocant mètode `existeixPaisBD`
    Si existeix el país, s'informa i finalitza
    Si no existeix el país, demana resta de dades (nom), crea objecte `Pais` i intenta afegir-lo a la BD invocant el mètode `afegirPaisBD`
    Si es produeix alguna excepció, s'informa del motiu.

Exercici per dijous, 23/11 (tasca 11)

- Ampliar classe `GestorBD` amb mètodes:
  - `public List<Pais> getPaisosBD(Connection con)`, que retorni la llista de països de la BD ordenats per codi iso.
  - `public Pais cercarPaisBD (Connection con, String codiISO)`, que cerqui el país dins la BD, retornant objecte `País` si existeix o `null` si no existeix

- A la classe `Programa`:
  - Programar mètode `mostrarPaisos` que invoqui `getPaisosBD`, a partir de la llista obtinguda, mostrar els països per la consola (si n'hi ha) o informar si no hi ha cap país.
  - Programar mètode `cercarPais`
    Ha de demanar un codi de país –controlant que sigui vàlid i transformant-lo en majúscules-.
    Ha de comprovar si existeix o el país a la BD, invocant mètode `cercarPaisBD`
    Si retorna `null`, s'informa de que no existeix.
    Si retorna el país, es mostra per pantalla.
    Si es produeix alguna excepció, s'informa del motiu.

Introducció als SBGDOO i accés des d'aplicacions	**21/11/23**

- Introducció als SGBDOO
  Veure document `DAMDAW_M03_UF6_introduccióAlsSGBDOO.pdf` adjunt.
- Introducció a un SGBDOO: Matisse
  Veure aquest vídeo
- Introducció al desenvolupament d'aplicacions que accedeixin a SGBDOO
  Veure aquest vídeo

Exercici per dimarts, 28/11 (tasca 12)

- Ampliar classe `GestorBD` amb mètodes:
  - `public boolean eliminarPaisBD(Connection con, String codiISO)` que intenti eliminar el país dins la BD, retornant `true` si l'ha pogut eliminar o `false` si no l'ha eliminat per què no existia.
  - `public void modificarPaisBD (Connection con, Pais p)` que intenti modificar el país dins la BD, generant excepció si a la BD no hi ha un país amb mateix identificador que el país p.

- A la classe `Programa`:
  - Programar mètode `eliminarPais`
    Ha de demanar un codi de país –controlant que sigui vàlid i transformant-lo en majúscules-.
    Ha d'intentar eliminar-lo de la BD invocant mètode `eliminarPaisBD`
    Si retorna `true`, s'informa de que s'ha eliminat.
    Si retorna `false`, s'informa de que no hi era.
    Si es produeix alguna excepció, s'informa del motiu.
  - Programar mètode `modificarPais`
    Ha de demanar un codi de país –controlant que sigui vàlid i transformant-lo en majúscules-.
    Ha de cercar el país a la BD, invocant mètode `cercarPaisBD` detallat abans.
    Si no existeix el país, s'informa i finalitza
    Si existeix s'ha de facilitar la possibilitat que modifiqui les dades del país, menys el seu identificador.
    En el cas de país que només té el camp `nom` que no és identificador, podria aparèixer un submenú com:
      1. Nom: <nom>
      0. Finalitzar

    i permet modificar les dades dels camps (en aquest cas només 1) tantes vegades com es desitgi fins escollir l'opció *Finalitzar*.
    En finalitzar ha d'enregistrar els canvis invocant mètode `modificarPaisBD`.
    Si es produeix alguna excepció, cal informar del motiu.

Exercici per dimarts, 5 de desembre (tasca 13)

- Ampliar projecte `P1_Model` amb la classe `Persona`, que ha de verificar:
  - `nif`: cadena(9) – obligatori amb 9 caràcters – majúscules
  - `cognomsNom`: cadena(60) – obligatori no buit
  - `dataNaix`: data – no obligatori i no futura
  - `pais`: referència a la classe `Pais`, no obligatori.
  - Mètodes habituals: constructor, getter-setter, `toString`
  - Considerar que 2 persones són iguals si tenen mateix `nif`

- Amplicar guió SQL on es creava la taula `PAIS` amb la creació de la taula `PERSONA` pensada per guardar les dades de persones de la classe `Persona`.

- Retocar el `main` de la classe `Programa` del projecte `P2_GestioBD` de manera que:
  - En iniciar, continuï establint connexió
  - Hi hagi un menú inicial amb 3 opcions:
    1. Gestió de països => Invoqui el menú ja existent amb opcions per gestionar països
    2. Gestió de persones => Invoqui nou menú per gestionar persones (opcions més avall).
    0. Finalitzar

- Opcions del nou menú per gestionar persones:
  1. Afegir persona => Ha d'invocar mètode `afegirPersona`
  2. Mostrar persones => Ha d'invocar mètode `mostrarPersones`
  3. Cercar persona per NIF => Ha d'invocar mètode `cercarPersona`
  4. Modificar persona => Ha d'invocar mètode `modificarPersona`
  5. Eliminar persona => Ha d'invocar mètode `eliminarPersona`
  6. Enregistrar canvis => Ha d'invocar mètode `validarCanvis` que ja tenim
  7. Desfer canvis => Ha d'invocar mètode `desferCanvis` que ja tenim
  0. Sortir

Inicialment, incorporeu tots els mètodes `xxxPersona` dins la classe `Programa` amb codi `/* TO DO */`

- Ampliar classe `GestorBD` amb mètodes següents, que en cas d'error han de generar una `GestorBDException` com en els mètodes ja desenvolupats. **MAI han de donar error per pantalla; no han de tenir interacció amb l'usuari**.
  - `public boolean existeixPersonaBD (Connection con, String nifPersona)`, que comprovi si existeix la persona indicada.
  - `public void afegirPersonaBD(Connection con, Persona p)`, que intenti inserir la persona a la BD, amb

- A la classe `Programa`:
  - Programar mètode `afegirPersona` :
    Ha de demanar un nif de persona –controlant que sigui vàlid i transformant-lo en majúscules-.
    Ha de comprovar si existeix o no a la BD, invocant mètode `existeixPersonaBD`.
    Si existeix la persona, s'informa i finalitza
    Si no existeix la persona, demana resta de dades...
    En referència al codi de país que l'usuari introdueixi, cal cercar el país a la BD amb el mètode `cercarPaisBD`.
    Si el país existeix, ja es pot crear l'objecte `Persona` i intentar afegir-lo a la BD invocant el mètode `afegirPersonaBD`.
    Si el país no existeix, cal informar a l'usuari preguntant si vol incorporar-lo.
      Si no el vol incorporar, es procedeix a avortar la introducció de la persona.
      En cas de voler-lo incorporar, es procedirà a demanar el nom del país per posteriorment crear els objectes `Pais` i `Persona`.
      Seguidament s'inserirà l'objecte `Pais` a la BD invocant el mètode `afegirPaisBD` i s'inserirà l'objecte `Persona` a la BD invocant el mètode `afegirPersonaBD`.
    Si es produeix alguna excepció, cal informar del motiu.

Exercici per dimarts, 12/12 (tasca 14)

- Ampliar classe `GestorBD` amb mètodes:
  - `public List<Persona> getPersonesBD(Connection con)`, que retorni la llista de persones de la BD ordenats per cognoms-nom. Si la persona té país, caldrà anar a cercar el país, cosa que cal fer invocant el mètode `cercarPaisBD` programat prèviament.
  - `public Pais cercarPersonaBD (Connection con, String NIF)`, que cerqui la persona dins la BD, retornant objecte `Persona` si existeix o `null` si no existeix. Si la persona té país, caldrà anar a cercar el país, cosa que cal fer invocant el mètode `cercarPaisBD` programat prèviament.
  - `public boolean eliminarPersonaBD(Connection con, String NIF)` que intenti eliminar la persona dins la BD, retornant `true` si l'ha pogut eliminar o `false` si no l'ha eliminat per què no existia.
  - `public void modificarPersonaBD (Connection con, Persona p)` que intenti modificar la persona dins la BD, generant excepció si a la BD no hi ha una persona amb mateix NIF que la persona `p`.

- A la classe `Programa`:
  - Programar mètode `mostrarPersones` que invoqui `getPersonesBD` i a partir de la llista obtinguda, mostrar les persones per la consola (si n'hi ha) o informar si no hi ha cap persona. Si la persona té país, mostrar el codi ISO i el nom.
  - Programar mètode `cercarPersona`
    Ha de demanar un NIF –controlant que sigui vàlid (obligatori 9 caràcters) i transformant-lo en majúscules-.
    Ha de comprovar si existeix la persona a la BD, invocant mètode `cercarPersonaBD`
    Si retorna `null`, s'informa de que no existeix.
    Si retorna la persona, es mostra per pantalla.
    Si es produeix alguna excepció, s'informa del motiu.
  - Programar mètode `eliminarPersona`
    Ha de demanar un NIF –controlant que sigui vàlid (obligatori 9 caràcters) i transformant-lo en majúscules-.
    Ha d'intentar eliminar-lo de la BD invocant mètode `eliminarPersonaBD`
    Si retorna `true`, s'informa de que s'ha eliminat.

Si retorna `false`, s'informa de que no hi era.
Si es produeix alguna excepció, s'informa del motiu.

- Programar mètode `modificarPersona`

Ha de demanar un NIF –controlant que sigui vàlid (obligatori 9 caràcters) i transformant-lo en majúscules-.
Ha de cercar la persona a la BD, invocant mètode `cercarPersonaBD` detallat abans.
Si no existeix la persona, s'informa i finalitza
Si existeix s'ha de facilitar la possibilitat que modifiqui les dades de la persona, menys el seu NIF.
En el cas de persona, podria aparèixer un submenú com:

1. Cognoms-Nom: `<cognomsNom>`
2. Data de naixement: `<dataNaix>`
3. País: `<codiISO>`
0. Finalitzar

i permet modificar les dades dels camps 1-3 tantes vegades com es desitgi fins escollir l'opció *Finalitzar*.
En finalitzar ha d'enregistrar els canvis invocant mètode `modificarPersonaBD`. En el cas del país, cal comprovar que el codi ISO introduït existeixi a la BD, usant el mètode `cercarPaisBD`.
Si es produeix alguna excepció, cal informar del motiu.